MPRI

Abstract Interpretation of Mobile Systems

Jérôme Feret
Département d'Informatique de l'École Normale Supérieure
INRIA, ÉNS, CNRS
http://www.di.ens.fr/~feret

January 7th, 2015

Overview

1. Overview
2. Mobile systems
3. Non standard semantics
4. Abstract Interpretation
5. Environment analyses
6. Occurrence counting analysis
7. Thread partitioning
8. Conclusion

Systèmes mobiles

Un ensemble de composants qui interagissent.
Ces interactions permettent de :

- synchroniser l'exécution de ces composants,
- changer les liaisons entre les composants,
- créer des nouveaux composants.

Le nombre de composants n'est pas borné !
Champs d'application :

- protocoles de communication,
- protocoles cryptographiques,
- systèmes reconfigurables,
- systèmes biologiques,
- ....

Démarche

Construction de sémantiques abstraites :

- correctes, automatiques, décidables,
- mais approchées (indécidabilité).

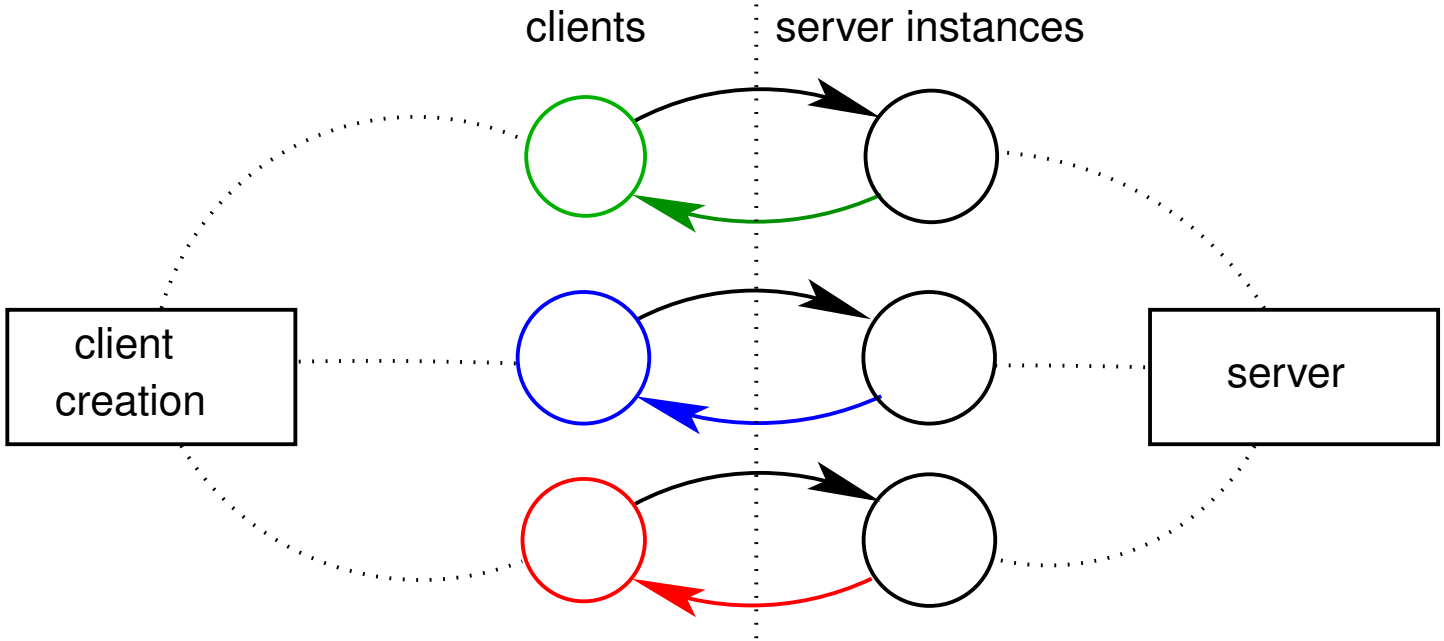
Approche indépendante du modèle:

1. conception d'un META-langage pour encoder les modèles existants ;
2. développement d'analyses au niveau de ce META-langage.

Trois familles d'analyses:

1. analyse des liaisons dynamiques entre les composants :
   (confinement, confidentialité, . . .)
2. dénombrement des composants :
   (exclusions mutuelles, non-épuisement des ressources, . . .)
3. analyse des unités de calculs :
   (absence de conflit, authentification, intégrité des mises à jour, . . .).

Analyse des liaisons entre les composants : Quels composants peuvent interagir ?

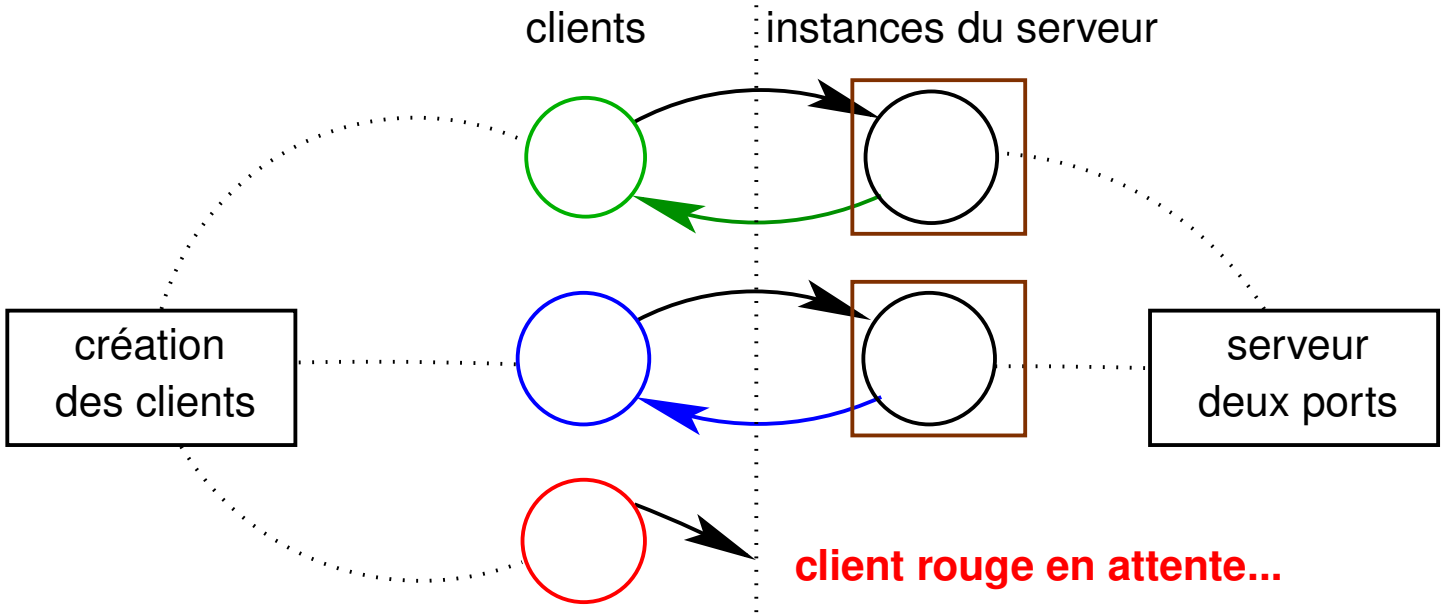


L'analyse distingue les composants récursifs

Domaines abstraits : relations entre des mots.

Publications : Feret — SAS'00, SAS'01, ESOP'02, JLAP'05

Analyse du nombre des composants : Borne le nombre de composants



Nouveau domaine abstrait : relations numériques
(invariants affines et intervalles).

Publication : Feret — GETCO'00, JLAP'05

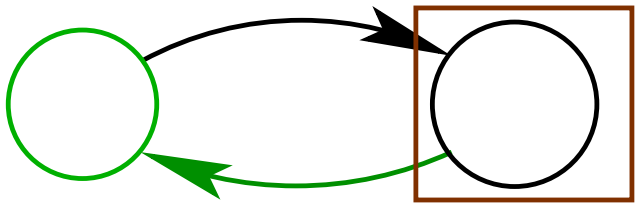
Partitionnement de tâches

Principe :

- regrouper les composants en unités de calcul,
  $\Longrightarrow$ grâce à l'analyse des liaisons entre les composants ;
- compter le nombre de composants dans chaque unité de calcul,
  $\Longrightarrow$ grâce à l'analyse de dénombrement.

Intérêt :

- chaque session est isolée,



ce qui permet aux analyses de se focaliser sur chacune des sessions.

Overview

1. Overview
2. Mobile systems
3. Non standard semantics
4. Abstract Interpretation
5. Environment analyses
6. Occurrence counting analysis
7. Thread partitioning
8. Conclusion

Mobile system

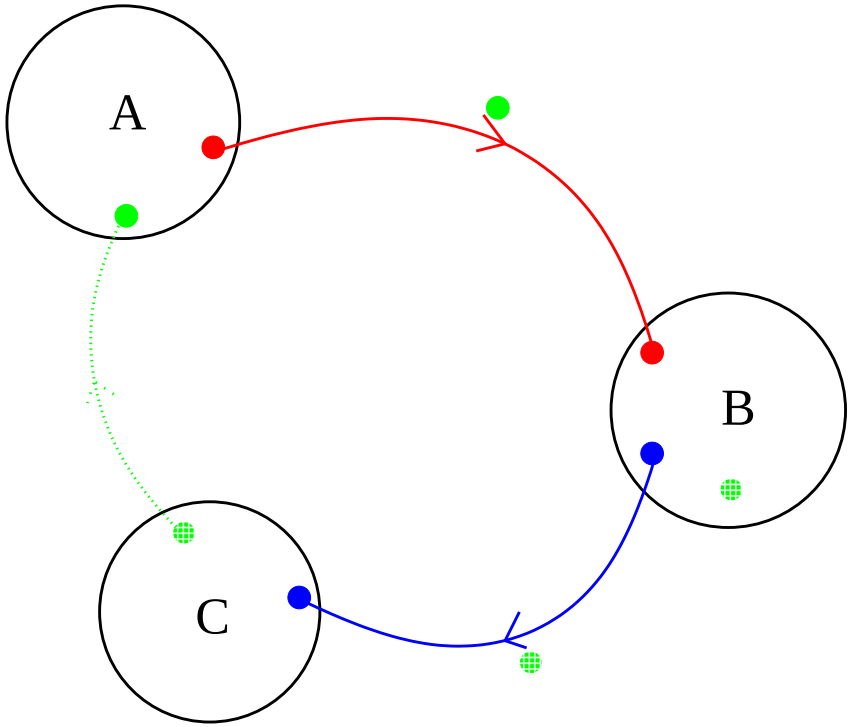
A pool of processes which interact and communicate:

Interactions control:

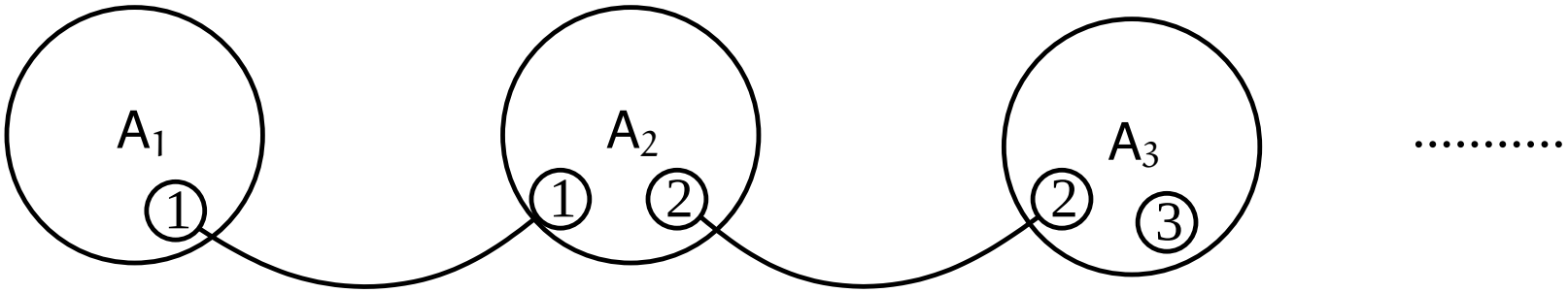
- process synchronization;
- update of link between processes (communication, migration);
- process creation.

The number of processes may be unbounded !

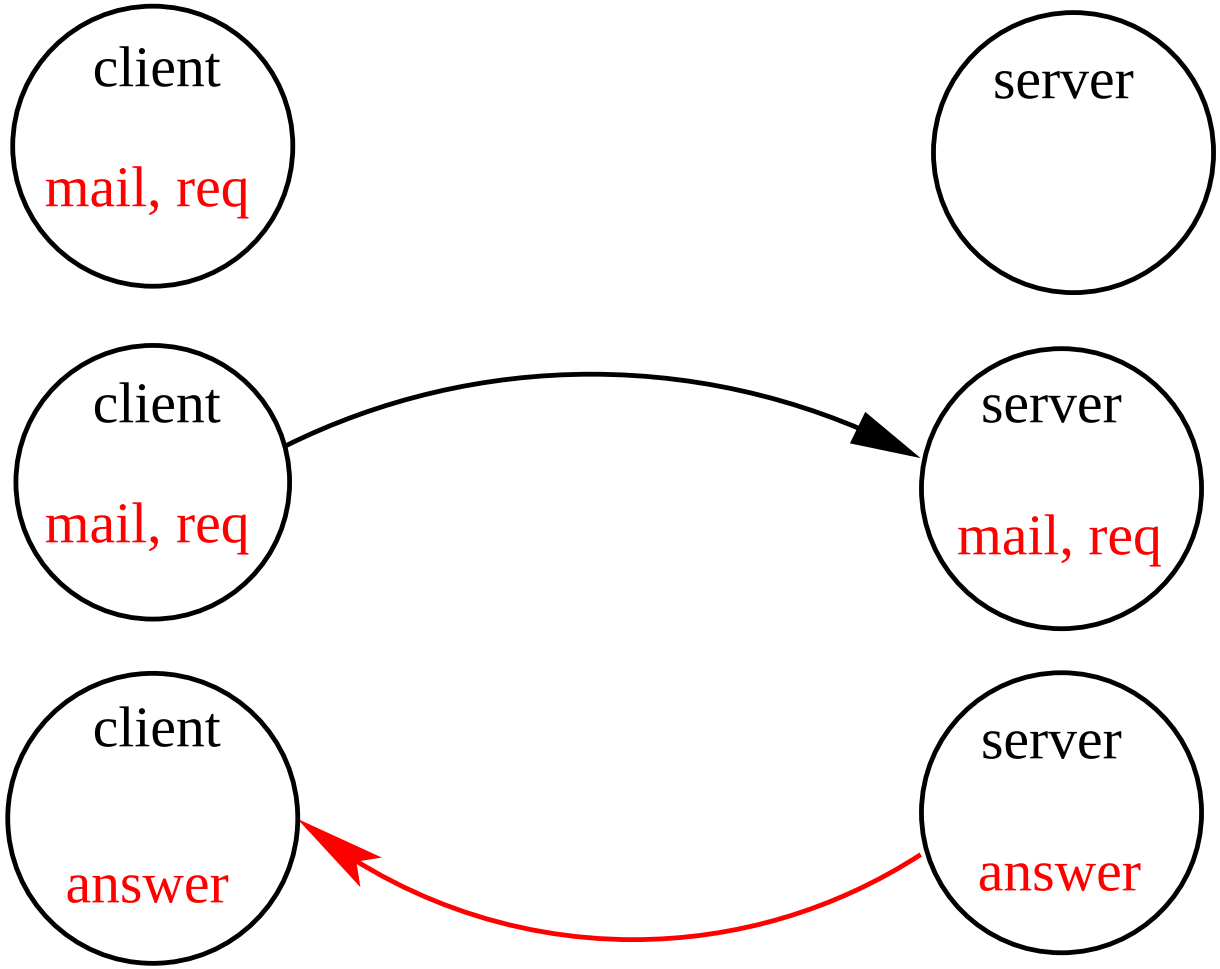
Dynamic linkage of agents



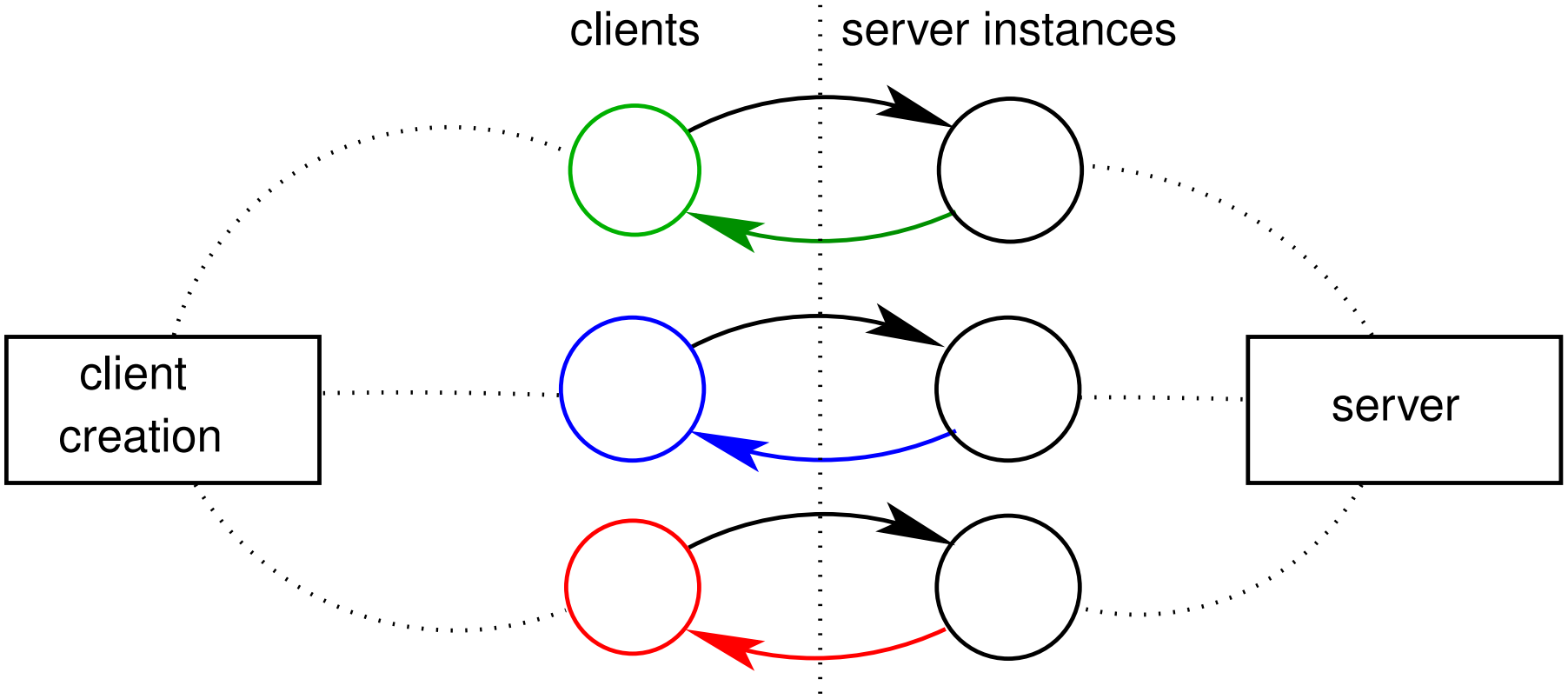
Dynamic creation of agents



A connection:



A network



π-calculus: syntax

Name: infinite set of channel names,
Label: infinite set of labels,

$$
\begin{array}{rcl}
P & ::= & \text{action}.P \\
 & | & (P \mid P) \\
 & | & (\nu\ x)P \\
 & | & \emptyset
\end{array}
$$

$$
\begin{array}{rcl}
\text{action} & ::= & c!^{i}[x_1, ..., x_n] \\
 & | & c?^{i}[x_1, ..., x_n] \\
 & | & *c?^{i}[x_1, ..., x_n]
\end{array}
$$

where $n \geqslant 0$, c, x_1, ..., x_n, x, $\in$ *Name*, $i \in$ *Label*.

ν and $?$ are the only name binders.
$\mathrm{fn}(P)$: free variables in P,
$\mathrm{bn}(P)$: bound names in P.

Transition semantics

A reduction relation and a congruence relation give the semantics of the π-calculus:

- the reduction relation specifies the result of computations:

$$c?^{i}[\overline{y}]Q \mid c!^{j}[\overline{x}]P \xrightarrow{i,j} Q[\overline{y} \leftarrow \overline{x}] \mid P$$

$$*c?^{i}[\overline{y}]Q \mid c!^{j}[\overline{x}]P \xrightarrow{i,j} Q[\overline{y} \leftarrow \overline{x}] \mid *c?^{i}[\overline{y}]Q \mid P$$

$$\frac{P \to Q}{(\nu\ x)P \to (\nu\ x)Q} \qquad \frac{P' \equiv P \quad P \to Q \quad Q \equiv Q'}{P' \to Q'} \qquad \frac{P \to P'}{P \mid Q \to P' \mid Q}$$

Congruence relation

- the congruence relation reveals redexs:

$$
\begin{array}{rcll}
P \mid Q & \equiv & Q \mid P & \text{(Commutativity)} \\
P \mid (Q \mid R) & \equiv & (P \mid Q) \mid R & \text{(Associativity)} \\
(\nu\, x)P & \equiv & (\nu\, y)P[x \leftarrow y] \quad \text{if } y \notin \mathrm{fn}(P) & (\alpha\text{-conversion}) \\
(\nu\, x)(\nu\, y)P & \equiv & (\nu\, y)(\nu\, x)P & \text{(Swapping)} \\
((\nu\, x)P) \mid Q & \equiv & (\nu\, x)(P \mid Q) \quad \text{if } x \notin \mathrm{fn}(Q) & \text{(Extrusion)} \\
(\nu\, x)\emptyset & \equiv & \emptyset & \text{(Garbage collection)}
\end{array}
$$

Exporting a channel

$(\nu\ a)((\nu\ x)(a?[y].P(x,y)|(\nu\ y)(\nu\ x)a![x].R(x,y)))$

$\equiv$ (α-conversion, swapping and extrusion)

$(\nu\ a)(\nu\ x_1)(\nu\ x_2)(\nu\ y)(a?[y].P(x_1,y)|a![x_2].R(x_2,y))$

$\rightarrow$

$(\nu\ a)(\nu\ x_1)(\nu\ x_2)(\nu\ y)(P(x_1,x_2)|R(x_2,y))$

$\equiv$ (swapping and extrusion)

$(\nu\ a)(\nu\ x_2)((\nu\ x_1)P(x_1,x_2)|(\nu\ y)R(x_2,y))$

Example: syntax

$$\mathcal{S} := (\nu\ \text{port})(\nu\ \text{gen})\ (\textbf{Server} \mid \textbf{Customer} \mid \text{gen}!^{0}[])$$

where

$$\textbf{Server} := *\text{port}?^{1}[\mathit{info},\mathit{add}](\mathit{add}!^{2}[\mathit{info}])$$

$$\textbf{Customer} := *\text{gen}?^{3}[]\ ((\nu\ \mathit{data})\ (\nu\ \mathit{email})\ (\text{port}!^{4}[\mathit{data},\mathit{email}] \mid \text{gen}!^{5}[]))$$

Example: computation

$$
\begin{array}{ll}
 & (\nu\ \text{port})(\nu\ \text{gen}) \\
 & \quad (\textbf{Server} \mid \textbf{Customer} \mid \text{gen}!^{0}[]) \\
\xrightarrow{3,0} & (\nu\ \text{port})(\nu\ \text{gen})(\nu\ \mathit{data}_1)(\nu\ \mathit{email}_1) \\
 & \quad (\textbf{Server} \mid \textbf{Customer} \mid \text{gen}!^{5}[] \mid \text{port}!^{4}[\mathit{data}_1, \mathit{email}_1]) \\
\xrightarrow{1,4} & (\nu\ \text{port})(\nu\ \text{gen})(\nu\ \mathit{data}_1)(\nu\ \mathit{email}_1) \\
 & \quad (\textbf{Server} \mid \textbf{Customer} \mid \text{gen}!^{5}[] \mid \mathit{email}_1!^{2}[\mathit{data}_1]) \\
\xrightarrow{3,5} & (\nu\ \text{port})(\nu\ \text{gen})(\nu\ \mathit{data}_1)(\nu\ \mathit{email}_1)(\nu\ \mathit{data}_2)(\nu\ \mathit{email}_2) \\
 & \quad (\textbf{Server} \mid \textbf{Customer} \mid \text{gen}!^{5}[] \mid \mathit{email}_1!^{2}[\mathit{data}_1] \mid \text{port}!^{4}[\mathit{data}_2, \mathit{email}_2]) \\
\xrightarrow{1,4} & (\nu\ \text{port})(\nu\ \text{gen})(\nu\ \mathit{data}_1)(\nu\ \mathit{email}_1)(\nu\ \mathit{data}_2)(\nu\ \mathit{email}_2) \\
 & \quad (\textbf{Server} \mid \textbf{Customer} \mid \text{gen}!^{5}[] \mid \mathit{email}_1!^{2}[\mathit{data}_1] \mid \mathit{email}_2!^{2}[\mathit{data}_2])
\end{array}
$$

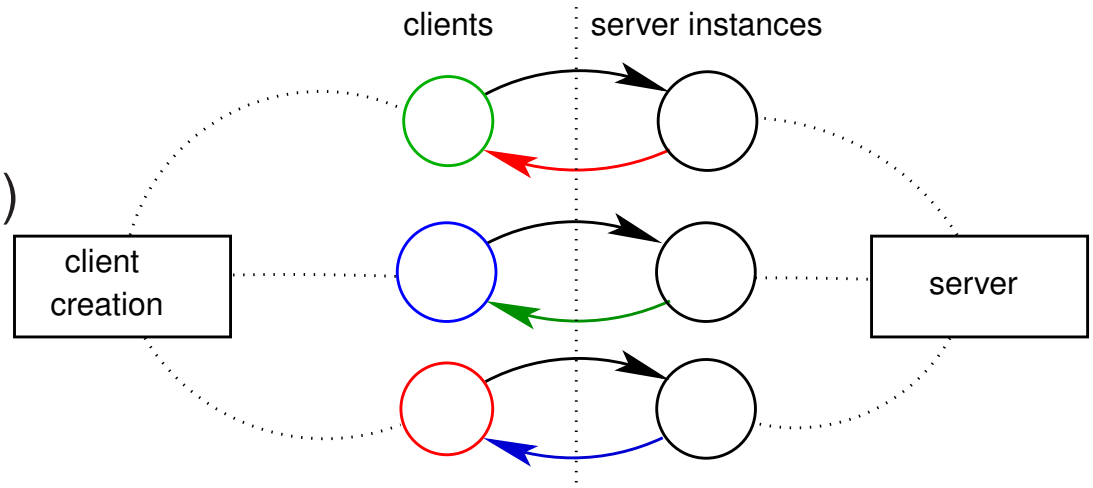
α-conversion

α-conversion destroys the link between names and processes which have declared them:

$$(\nu\ \text{port})(\nu\ \text{gen})(\nu\ \mathit{data}_1)(\nu\ \mathit{email}_1)$$
$$(\nu\ \mathit{data}_2)(\nu\ \mathit{email}_2)$$
$$(\textbf{Server} \mid \textbf{Customer} \mid \text{gen}!^5[]$$
$$\mid \mathit{email}_1!^4[\mathit{data}_1] \mid \mathit{email}_2!^4[\mathit{data}_2])$$

$\sim_\alpha$

$$(\nu\ \text{port})(\nu\ \text{gen})(\nu\ \mathit{data}_2)$$
$$(\nu\ \mathit{email}_1)(\nu\ \mathit{data}_1)(\nu\ \mathit{email}_2)$$
$$(\textbf{Server} \mid \textbf{Customer} \mid \text{gen}!^5[]$$
$$\mid \mathit{email}_1!^4[\mathit{data}_2] \mid \mathit{email}_2!^4[\mathit{data}_1])$$



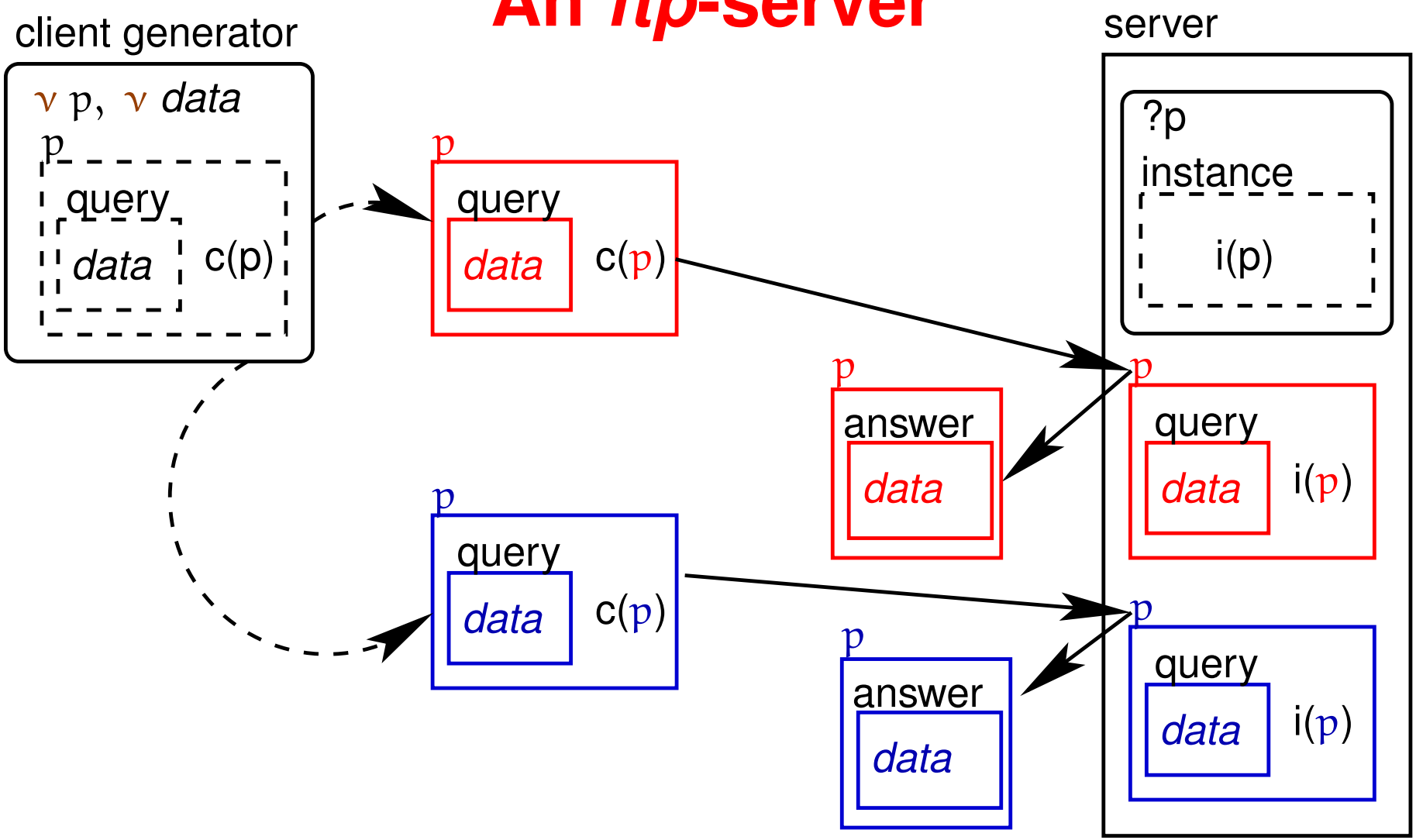
Mobile Ambients

Ambients are named boxes containing other ambients (and/or) some agents.

Agents:

- provide capabilities to their surrounding ambients for local migration and other ambient dissolution;
- dynamically create new ambients, names and agents;
- communicate names to each others.

An *ftp*-server



Syntax

Let *Name* be an infinite countable set of ambient names and *Label* an infinite countable set of labels.

$$
\begin{array}{rcll}
n & \in & \mathit{Name} & \text{(ambient name)} \\
l & \in & \mathit{Label} & \text{(label)}
\end{array}
$$

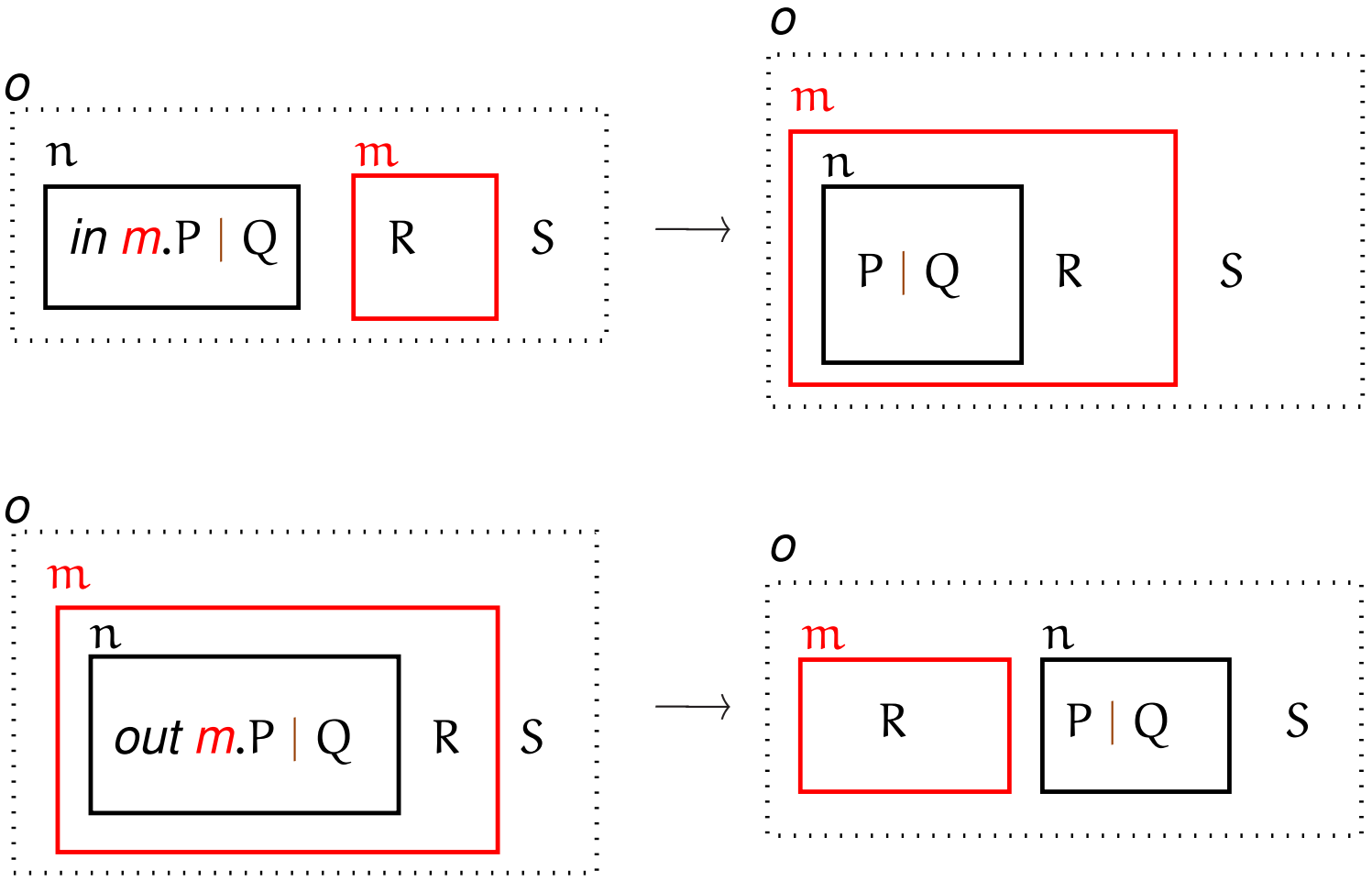
$$
\begin{array}{rcll}
P, Q & ::= & (\nu\ n)P & \text{(restriction)} \\
 & | & \mathbf{0} & \text{(inactivity)} \\
 & | & P \mid Q & \text{(composition)} \\
 & | & n^{l}[P] & \text{(ambient)} \\
 & | & M & \text{(capability action)} \\
 & | & \mathit{io} & \text{(input/output action)}
\end{array}
$$

Capability and actions

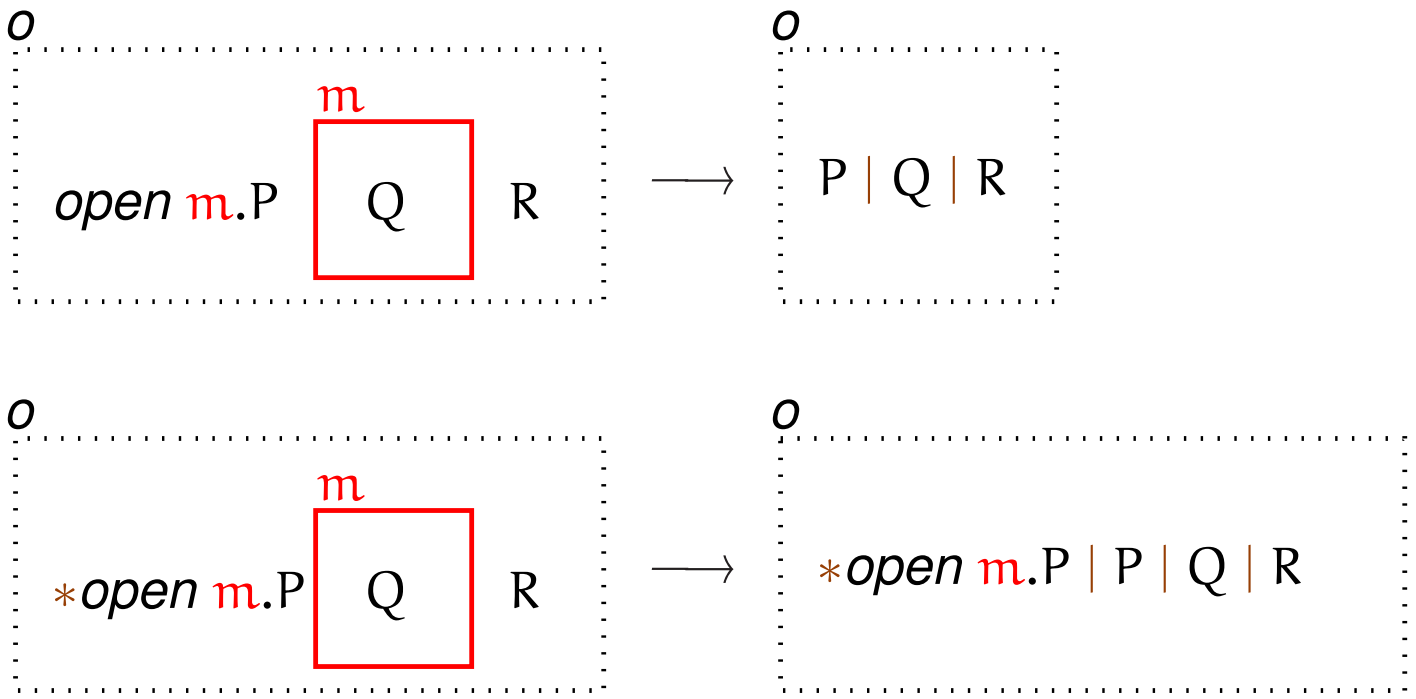
$$
\begin{array}{rcll}
M & ::= & in^{l}\ n.P & \text{(can enter an ambient named } n\text{)} \\
 & | & out^{l}\ n.P & \text{(can exit an ambient named } n\text{)} \\
 & | & open^{l}\ n.P & \text{(can open an ambient named } n\text{)} \\
 & | & !open^{l}\ n.P & \text{(can open several ambients named } n\text{)} \\
 & & & \\
io & ::= & (n)^{l}.P & \text{(input action)} \\
 & | & !(n)^{l}.P & \text{(input action with replication)} \\
 & | & \langle n \rangle^{l} & \text{(async output action)}
\end{array}
$$

The only name binders are $(\nu\ \_)$, $(\_)$ and $!(\_)$.

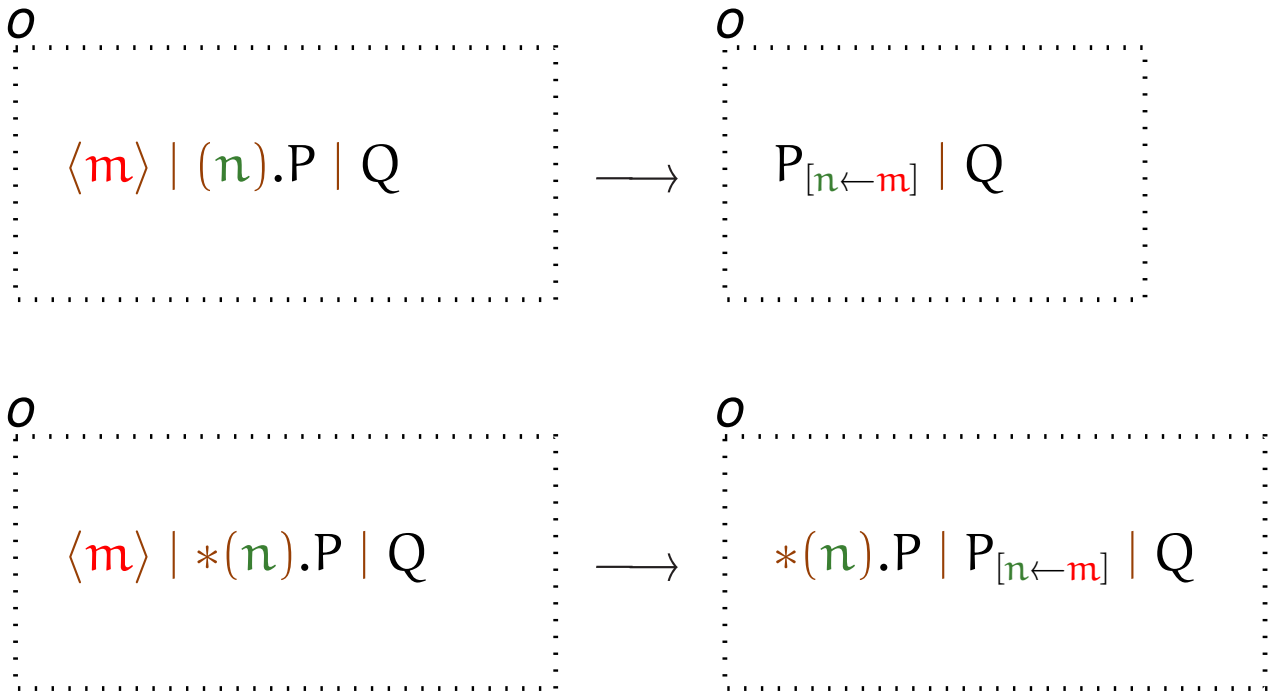
Ambient Migration



Ambient Dissolution



Communication



An *ftp*-server

$\mathcal{S} := (\nu \mathbf{Pub})(\mathbf{S} \mid !(x)^{11}.C \mid \langle \mathsf{make} \rangle^{21})$

where

$\mathbf{Pub} := (\nu\ \mathsf{request})(\nu\ \mathsf{make})(\nu\ \mathsf{server})(\nu\ \mathsf{duplicate})(\nu\ \mathsf{instance})(\nu\ \mathsf{answer}),$

$\mathbf{C} := (\nu\ q)(\nu\ p)p^{12}[\mathbf{C}_1 \mid \mathbf{C}_2 \mid \mathbf{C}_3] \mid \langle \mathsf{make} \rangle^{20},$
$\mathbf{C}_1 := \mathsf{request}^{13}[\langle q \rangle^{14}],\ \mathbf{C}_2 := \mathit{open}^{15}\mathsf{instance},$
$\mathbf{C}_3 := \mathit{in}^{16}\mathsf{server.duplicate}^{17}[\mathit{out}^{18}p.\langle p \rangle^{19}],$

$\mathbf{S} := \mathsf{server}^{1}[\mathbf{S}_1 \mid \mathbf{S}_2],\ \mathbf{S}_1 := !\mathit{open}^{2}\mathsf{duplicate},\ \mathbf{S}_2 := !(k)^{3}.\mathsf{instance}^{4}[\mathbf{I}],$

$\mathbf{I} := \mathit{in}^{5}k.\mathit{open}^{6}\mathsf{request}.(\mathit{rep})^{7}(\mathbf{I}_1 \mid \mathbf{I}_2),\ \mathbf{I}_1 := \mathsf{answer}^{8}[\langle \mathit{rep} \rangle^{9}],\ \mathbf{I}_2 := \mathit{out}^{10}\mathsf{server}.$

$$
\begin{array}{l}
\quad (\nu\mathbf{Pub})(\mathbf{S} \mid !(x)^{11}.C \mid \langle \mathsf{make} \rangle^{21}) \\
\longrightarrow \\
\quad (\nu\mathbf{Pub}) \left(\begin{array}{l} !(x)^{11}.C \mid \langle \mathsf{make} \rangle^{20} \mid \mathsf{server}^{1}[\mathbf{S}_1 \mid \mathbf{S}_2] \mid \\ (\nu\ q_1)(\nu\ p_1) p_1{}^{12} \left[\begin{array}{l} \mathsf{request}^{13}[\langle q_1 \rangle^{14}] \mid \mathbf{C}_2 \mid \\ \mathit{in}^{16}\mathsf{server}.\mathsf{duplicate}^{17}[\mathit{out}^{18} p_1.\langle p_1 \rangle^{19}] \end{array} \right] \end{array} \right) \\
\longrightarrow \\
\quad (\nu\mathbf{Pub})(\nu\ q_1)(\nu\ p_1) \\
\quad \left(!(x)^{11}.C \mid \langle \mathsf{make} \rangle^{20} \mid \mathsf{server}^{1} \left[\mathbf{S}_1 \mid \mathbf{S}_2 \mid p_1{}^{12} \left[\begin{array}{l} \mathsf{request}^{13}[\langle q_1 \rangle^{14}] \mid \mathbf{C}_2 \mid \\ \mathsf{duplicate}^{17}[\mathit{out}^{18} p_1.\langle p_1 \rangle^{19}] \end{array} \right] \right] \right) \\
\longrightarrow \\
\quad (\nu\mathbf{Pub})(\nu\ q_1)(\nu\ p_1) \\
\quad \left(!(x)^{11}.C \mid \langle \mathsf{make} \rangle^{20} \mid \mathsf{server}^{1} \left[\begin{array}{l} !\mathit{open}^{2}\mathsf{duplicate} \mid \mathbf{S}_2 \mid \mathsf{duplicate}^{17}[\langle p_1 \rangle^{19}] \mid \\ p_1{}^{12}[\mathsf{request}^{13}[\langle q_1 \rangle^{14}] \mid \mathbf{C}_2] \end{array} \right] \right)
\end{array}
$$

$$(\nu\mathbf{Pub})(\nu\ q_1)(\nu\ p_1)\left(!(x)^{11}.C \mid \langle \mathsf{make}\rangle^{20} \mid \mathsf{server}^1\left[\begin{array}{l} !\mathit{open}^2\mathsf{duplicate} \mid \mathbf{S}_2 \mid \mathsf{duplicate}^{17}[\langle p_1\rangle^{19}] \mid \\ p_1{}^{12}[\mathsf{request}^{13}[\langle q_1\rangle^{14}] \mid \mathbf{C}_2] \end{array}\right]\right)$$

$\longrightarrow$

$$(\nu\mathbf{Pub})(\nu\ q_1)(\nu\ p_1)\left(!(x)^{11}.C \mid \langle \mathsf{make}\rangle^{20} \mid \mathsf{server}^1\left[\begin{array}{l} \mathbf{S}_1 \mid !(k)^3.\mathsf{instance}^4[\mathbf{I}] \mid \langle p_1\rangle^{19} \mid \\ p_1{}^{12}[\mathsf{request}^{13}[\langle q_1\rangle^{14}] \mid \mathbf{C}_2] \end{array}\right]\right)$$

$\longrightarrow$

$$(\nu\mathbf{Pub})(\nu\ q_1)(\nu\ p_1)\left(!(x)^{11}.C \mid \langle \mathsf{make}\rangle^{20} \mid \mathsf{server}^1\left[\begin{array}{l} \mathbf{S}_1 \mid \mathbf{S}_2 \mid p_1{}^{12}[\mathsf{request}^{13}[\langle q_1\rangle^{14}] \mid \mathbf{C}_2] \mid \\ \mathsf{instance}^4[\mathit{in}^5 p_1.\mathit{open}^6\mathsf{request}.(\mathit{rep})^7(\mathbf{I}_1|\mathbf{I}_2)] \end{array}\right]\right)$$

$\longrightarrow$

$$(\nu\mathbf{Pub})(\nu\ q_1)(\nu\ p_1)\left(!(x)^{11}.C \mid \langle \mathsf{make}\rangle^{20} \mid \mathsf{server}^1\left[\begin{array}{l} \mathbf{S}_1 \mid \mathbf{S}_2 \mid \\ p_1{}^{12}\left[\begin{array}{l} \mathsf{request}^{13}[\langle q_1\rangle^{14}] \mid \mathit{open}^{15}\mathsf{instance} \mid \\ \mathsf{instance}^4[\mathit{open}^6\mathsf{request}.(\mathit{rep})^7(\mathbf{I}_1|\mathbf{I}_2)] \end{array}\right] \end{array}\right]\right)$$

$$
\begin{array}{l}
\quad (\nu\,\mathbf{Pub})(\nu\ q_1)(\nu\ p_1) \\
\quad \left(!(x)^{11}.C \mid \langle \mathsf{make} \rangle^{20} \mid \mathsf{server}^{1} \left[\begin{array}{l} \mathbf{S}_1 \mid \mathbf{S}_2 \mid \\ p_1{}^{12} \left[\begin{array}{l} \mathsf{request}^{13}[\langle q_1 \rangle^{14}] \mid \mathit{open}^{15}\mathsf{instance} \mid \\ \mathsf{instance}^{4}[\mathit{open}^{6}\mathsf{request}.(\mathit{rep})^{7}(\mathsf{I}_1 \mid \mathsf{I}_2)] \end{array} \right] \end{array} \right] \right) \\
\longrightarrow \\
\quad (\nu\,\mathbf{Pub})(\nu\ q_1)(\nu\ p_1) \\
\quad \left(!(x)^{11}.C \mid \langle \mathsf{make} \rangle^{20} \mid \mathsf{server}^{1} \left[\mathbf{S}_1 \mid \mathbf{S}_2 \mid p_1{}^{12} \left[\begin{array}{l} \mathsf{request}^{13}[\langle q_1 \rangle^{14}] \mid \\ \mathit{open}^{6}\mathsf{request}.(\mathit{rep})^{7}(\mathsf{I}_1 \mid \mathsf{I}_2) \end{array} \right] \right] \right) \\
\longrightarrow^{*} \\
\quad (\nu\,\mathbf{Pub})(\nu\ q_1)(\nu\ p_1) \\
\quad (!(x)^{11}.C \mid \langle \mathsf{make} \rangle^{20} \mid \mathsf{server}^{1}[\mathbf{S}_1 \mid \mathbf{S}_2 \mid p_1{}^{12}[\mathsf{answer}^{8}[\langle q_1 \rangle^{9}] \mid \mathit{out}^{10}\mathsf{server}]]) \\
\longrightarrow \\
\quad (\nu\,\mathbf{Pub})(\nu\ q_1)(\nu\ p_1) \\
\quad (!(x)^{11}.C \mid \langle \mathsf{make} \rangle^{20} \mid \mathsf{server}^{1}[\mathbf{S}_1 \mid \mathbf{S}_2] \mid p_1{}^{12}[\mathsf{answer}^{8}[\langle q_1 \rangle^{9}]]) \\
\longrightarrow^{*} \\
\quad (\nu\,\mathbf{Pub})(\nu\ q_1)(\nu\ p_1)(\nu\ q_2)(\nu\ p_2)(!(x)^{11}.C \mid \langle \mathsf{make} \rangle^{20} \mid \mathsf{server}^{1}[\mathbf{S}_1 \mid \mathbf{S}_2] \\
\qquad\qquad\qquad\qquad \mid p_1{}^{12}[\mathsf{answer}^{8}[\langle q_1 \rangle^{9}]] \mid p_2{}^{12}[\mathsf{answer}^{8}[\langle q_2 \rangle^{9}]])
\end{array}
$$

Shared-memory example
Motivation

We want to describe in the π-calculus a shared-memory in which:

- each process can allocate new cells,
- each authorized process can read the content of a cell,
- each authorized process can write inside a cell, overwriting the former content.

Shared-memory example
Specification

A memory cell will be denoted by three channel names, *cell*, *read*, *write*:

- a channel name *cell* describes the content of the cell:
  the process $cell![data]$ means that the cell *cell* contains the information *data*, this name is internal to the memory (not visible by the user).
- a channel name *read* allows reading requests:
  the process $read![port]$ is a request to read the content of the cell, and send it to the port *port*,
- a channel name *write* allows writing requests:
  the process $write![data]$ is a request to write the information *data* inside the cell.

Shared Memory Encoding

System := (ν create)(ν null)($*$create?[d].Allocate(d))

Allocate(d) :=
(ν *cell*)(ν *write*)(ν *read*)
(init(*cell*) | read(*read*,*cell*) | write(*write*,*cell*) | d![*read*;*write*])

where

- init(*cell*) := *cell*![null]
- read(*read*,*cell*) := $*$*read*?[*port*].*cell*?[*u*].(*cell*![*u*] | *port*![*u*])
- write(*write*,*cell*) := $*$*write*?[*data*].*cell*?[*u*].*cell*![*data*]

Shared-memory example
Trace example

$(\nu\ \text{create})(\nu\ \text{null})$
$\quad (*\text{create}?[d].\text{Allocate}(d)$
$\quad\ \mid (\nu\ \mathit{address})(\nu\ \mathit{data})\text{create}![\mathit{address}].\mathit{address}?[r;w].w![\mathit{data}].r![\mathit{address}])$
$\longrightarrow$
$(\nu\ \text{create})\ (\nu\ \text{null})\ (\nu\ \mathit{cell})\ (\nu\ \mathit{write})\ (\nu\ \mathit{read})\ (\nu\ \mathit{address})\ (\nu\ \mathit{data})$
$\quad (\ *\text{create}?[d].\text{Allocate}(d)\ \mid\ \text{read}(\mathit{read},\mathit{cell})\ \mid\ \text{write}(\mathit{write},\mathit{cell})$
$\quad\ \mid\ \mathit{cell}![\text{null}]\ \mid\ \mathit{address}![\mathit{read},\mathit{write}]\ \mid\ \mathit{address}?[r;w].w![\mathit{data}].r![\mathit{address}])$
$\longrightarrow$
$(\nu\ \overline{c})(\ *\text{create}?[d].\text{Allocate}(d)\ \mid\ \text{read}(\mathit{read},\mathit{cell})\ \mid\ \text{write}(\mathit{write},\mathit{cell})$
$\quad\ \mid\ \mathit{cell}![\text{null}]\ \mid\ \mathit{write}![\mathit{data}].\mathit{read}![\mathit{address}])$
$\longrightarrow$
$(\nu\ \overline{c})(\ *\text{create}?[d].\text{Allocate}(d)\ \mid\ \text{read}(\mathit{read},\mathit{cell})\ \mid\ \text{write}(\mathit{write},\mathit{cell})$
$\quad\ \mid\ \mathit{cell}![\text{null}]\ \mid\ \mathit{cell}?[u].\mathit{cell}![\mathit{data}]\mid \mathit{read}![\mathit{address}])$

Shared-memory example
Expected Derivation

$(\nu\ \overline{c})(\ *\text{create?}[d].\text{Allocate}(d)\ \mid\ \text{read}(\textit{read},\textit{cell})\ \mid\ \text{write}(\textit{write},\textit{cell})$
$\quad\mid\ \textit{cell}![\text{null}]\ \mid\ \textit{cell}?[u].\textit{cell}![\textit{data}]\ \mid \textit{read}![\textit{address}])$

$\longrightarrow$

$(\nu\ \overline{c})(\ *\text{create?}[d].\text{Allocate}(d)\ \mid\ \text{read}(\textit{read},\textit{cell})\ \mid\ \text{write}(\textit{write},\textit{cell})$
$\quad\mid\ \textit{cell}![\textit{data}] \mid \textit{read}![\textit{address}])$

$\longrightarrow$

$(\nu\ \overline{c})(\ *\text{create?}[d].\text{Allocate}(d)\ \mid\ \text{read}(\textit{read},\textit{cell})\ \mid\ \text{write}(\textit{write},\textit{cell})$
$\quad\mid\ \textit{cell}![\textit{data}] \mid \textit{cell}?[\text{u}].(\textit{cell}![\text{u}] \mid \textit{address}![\text{u}]))$

$\longrightarrow$

$(\nu\ \overline{c})(\ *\text{create?}[d].\text{Allocate}(d)\ \mid\ \text{read}(\textit{read},\textit{cell})\ \mid\ \text{write}(\textit{write},\textit{cell})$
$\quad\mid\ \textit{cell}![\textit{data}] \mid \textit{address}![\textit{data}])$

Shared-memory example
Unexpected Derivation

$(\nu\ \overline{c})(\ *\text{create?}[d].\text{Allocate}(d)\ \mid\ \text{read}(\textit{read},\textit{cell})\ \mid\ \text{write}(\textit{write},\textit{cell})$
$\quad\mid\ \textit{cell}![\text{null}]\ \mid\ \textit{cell}?[u].\textit{cell}![\textit{data}] \mid \textit{read}![\textit{address}])$

$\longrightarrow$

$(\nu\ \overline{c})(\ *\text{create?}[d].\text{Allocate}(d)\ \mid\ \text{read}(\textit{read},\textit{cell})\ \mid\ \text{write}(\textit{write},\textit{cell})$
$\quad\mid\ \textit{cell}![\text{null}]\ \mid\ \textit{cell}?[\text{u}].\textit{cell}![\textit{data}] \mid \textit{cell}?[\text{u}].(\textit{cell}![\text{u}] \mid \textit{address}![\text{u}]))$

$\longrightarrow$

$(\nu\ \overline{c})(\ *\text{create?}[d].\text{Allocate}(d)\ \mid\ \text{read}(\textit{read},\textit{cell})\ \mid\ \text{write}(\textit{write},\textit{cell})$
$\quad\mid\ \textit{cell}![\text{null}]\ \mid\ \textit{cell}?[\text{u}].\textit{cell}![\textit{data}] \mid \textit{address}![\text{null}])$

$\longrightarrow$

$(\nu\ \overline{c})(\ *\text{create?}[d].\text{Allocate}(d)\ \mid\ \text{read}(\textit{read},\textit{cell})\ \mid\ \text{write}(\textit{write},\textit{cell})$
$\quad\mid\ \textit{cell}![\textit{data}] \mid \textit{address}![\text{null}])$

Shared-memory example
Enforcing synchronisation

$$\mathrm{System} := (\nu\ \mathrm{create})(\nu\ \mathrm{null})(*\mathrm{create}?[\mathrm{d}].\mathrm{Allocate}(d))$$

$$\mathrm{Allocate}(\mathrm{d}) := (\nu\ \mathit{cell})(\nu\ \mathit{write})(\nu\ \mathit{read})\ \mathrm{init}(\mathit{cell}) \mid \mathrm{read}(\mathit{read},\mathit{cell}) \mid \mathrm{write}(\mathit{write},\mathit{cell}) \mid \mathrm{d}![\mathit{read};\mathit{write}]$$

where

- $\mathrm{init}(\mathit{cell}) := \mathit{cell}![\mathrm{null}]$
- $\mathrm{read}(\mathit{read},\mathit{cell}) := *\mathit{read}?[\mathit{port}].\mathit{cell}?[u](\mathit{cell}![u] \mid \mathit{port}![u])$
- $\mathrm{write}(\mathit{write},\mathit{cell}) := *\mathit{write}?[\mathit{data},\mathit{ack}].\mathit{cell}?[u].(\mathit{cell}![\mathit{data}] \mid \mathit{ack}![])$

$$
\begin{array}{l}
(\nu\ \mathrm{create})(\nu\ \mathrm{null}) \\
\quad (*\mathrm{create}?[d].\mathrm{Allocate}(d) \\
\quad\ \mid (\nu\ \mathit{address})(\nu\ \mathit{data})(\nu\ \mathit{ack}) \\
\qquad\quad \mathrm{create}![\mathit{address}].\mathit{address}?[r;w].w![\mathit{data};\mathit{ack}].\mathit{ack}?[].r![\mathit{address}]) \\
\longrightarrow \\
(\nu\ \overline{c})(\ *\mathrm{create}?[d].\mathrm{Allocate}(d)\ \mid\ \mathrm{read}(\mathit{read},\mathit{cell})\ \mid\ \mathrm{write}(\mathit{write},\mathit{cell})\ \mid\ \mathit{cell}![\mathrm{null}] \\
\quad\ \mid\ \mathit{address}![\mathit{read},\mathit{write}]\ \mid\ \mathit{address}?[r;w].w![\mathit{data};\mathit{ack}].\mathit{ack}?[].r![\mathit{address}]) \\
\longrightarrow \\
(\nu\ \overline{c})(\ *\mathrm{create}?[d].\mathrm{Allocate}(d)\ \mid\ \mathrm{read}(\mathit{read},\mathit{cell})\ \mid\ \mathrm{write}(\mathit{write},\mathit{cell}) \\
\qquad\ \mid\ \mathit{cell}![\mathrm{null}]\ \mid\ \mathit{write}![\mathit{data};\mathit{ack}].\mathit{ack}?[].\mathit{read}![\mathit{address}]) \\
\longrightarrow \\
(\nu\ \overline{c})(\ *\mathrm{create}?[d].\mathrm{Allocate}(d)\ \mid\ \mathrm{read}(\mathit{read},\mathit{cell})\ \mid\ \mathrm{write}(\mathit{write},\mathit{cell}) \\
\qquad\ \mid\ \mathit{cell}![\mathrm{null}]\ \mid\ \mathit{cell}?[u].(\mathit{cell}![\mathit{data}]\ \mid\ \mathit{ack}![]) \\
\qquad\ \mid\ \mathit{ack}?[].\mathit{read}![\mathit{address}])
\end{array}
$$

$$
\begin{array}{l}
(\nu\ \overline{c})(\ *\mathrm{create}?[d].\mathrm{Allocate}(d)\ \mid\ \mathrm{read}(\mathit{read},\mathit{cell})\ \mid\ \mathrm{write}(\mathit{write},\mathit{cell}) \\
\qquad \mid\ \mathit{cell}![\mathrm{null}]\ \mid\ \mathit{cell}?[u].(\mathit{cell}![\mathit{data}] \mid \mathit{ack}![]) \\
\qquad \mid \mathit{ack}?[].\mathit{read}![\mathit{address}]) \\
\longrightarrow \\
(\nu\ \overline{c})(\ *\mathrm{create}?[d].\mathrm{Allocate}(d)\ \mid\ \mathrm{read}(\mathit{read},\mathit{cell})\ \mid\ \mathrm{write}(\mathit{write},\mathit{cell}) \\
\qquad \mid\ (\mathit{cell}![\mathit{data}] \mid \mathit{ack}![]) \mid \mathit{ack}?[].\mathit{read}![\mathit{address}]) \\
\longrightarrow \\
(\nu\ \overline{c})(\ *\mathrm{create}?[d].\mathrm{Allocate}(d)\ \mid\ \mathrm{read}(\mathit{read},\mathit{cell})\ \mid\ \mathrm{write}(\mathit{write},\mathit{cell}) \\
\qquad \mid\ \mathit{cell}![\mathit{data}] \mid \mathit{read}![\mathit{address}]) \\
\longrightarrow \\
(\nu\ \overline{c})(\ *\mathrm{create}?[d].\mathrm{Allocate}(d)\ \mid\ \mathrm{read}(\mathit{read},\mathit{cell})\ \mid\ \mathrm{write}(\mathit{write},\mathit{cell}) \\
\qquad \mid\ \mathit{cell}![\mathit{data}] \mid\ \mathit{address}![\mathit{data}])
\end{array}
$$

Shared-memory example
Using Mutex

$$\mathsf{System} := (\nu\ \mathsf{create})(\nu\ \mathsf{null})(*\mathsf{create}?[d]\mathsf{Allocate}(d))$$
$$\mathsf{Allocate}(d) := (\nu\ \mathit{cell})(\nu\ \mathit{mutex})(\nu\ \mathit{nomutex})(\nu\ \mathit{write})(\nu\ \mathit{read})(\nu\ \mathit{lock})(\nu\ \mathit{unlock})$$
$$\mathsf{init}(\mathit{cell},\mathit{mutex}) \mid \mathsf{read}(\mathit{read},\mathit{cell}) \mid \mathsf{write}(\mathit{write},\mathit{cell})$$
$$\mid \mathsf{lock}(\mathit{lock},\mathit{mutex},\mathit{nomutex}) \mid \mathsf{unlock}(\mathit{unlock},\mathit{mutex},\mathit{nomutex})$$
$$\mid d![\mathit{read};\mathit{write};\mathit{lock};\mathit{unlock}]$$

where

$$\mathsf{init}(\mathit{cell},\mathit{mutex}) := \mathit{cell}![\mathsf{null}] \mid \mathit{mutex}![]$$
$$\mathsf{read}(\mathit{read},\mathit{cell}) := *\mathit{read}?[\mathit{port}].\mathit{cell}?[u](\mathit{cell}![u] \mid \mathit{port}![u])$$
$$\mathsf{write}(\mathit{write},\mathit{cell}) := *\mathit{write}?[\mathit{data},\mathit{ack}].\mathit{cell}?[u].(\mathit{cell}![\mathit{data}] \mid \mathit{ack}![])$$
$$\mathsf{lock}(\mathit{lock},\mathit{mutex},\mathit{nomutex}) := *\mathit{lock}?[\mathit{ack}].\mathit{mutex}?[].(\mathit{ack}![] \mid \mathit{nomutex}![])$$
$$\mathsf{unlock}(\mathit{unlock},\mathit{mutex},\mathit{nomutex}) := *\mathit{unlock}?[\mathit{ack}].\mathit{nomutex}?[].(\mathit{ack}![] \mid \mathit{mutex}![])$$

Overview

1. Overview
2. Mobile systems
3. Non standard semantics
4. Abstract Interpretation
5. Environment analyses
6. Occurrence counting analysis
7. Thread partitioning
8. Conclusion

Motivation

We focus on reachability properties.
We distinguish between recursive instances of components.

We design three families of analyses:

1. environment analyses capture dynamic properties
   (non-uniform control flow analysis, secrecy, confinement, ...)
2. occurrence counting captures concurrency properties
   (mutual exclusion, non exhaustion of resources)
3. thread partitioning mixes both dynamic and concurrency properties
   (absence of race condition, authentication, ...).

Non-standard semantics

A refined semantics in where

- recursive instances of processes are identified with unambiguous markers;
- channel names are stamped with the marker of the process which has declared them.

Example: non-standard configuration

$$(\textbf{Server} \mid \textbf{Client} \mid \text{gen}!^{5}[] \mid \mathit{email}_1!^{2}[\mathit{data}_1] \mid \mathit{email}_2!^{2}[\mathit{data}_2])$$

$$\left\{ \begin{array}{l} \left(1, \varepsilon, \left\{ \text{port} \mapsto (\text{port}, \varepsilon) \right.\right) \\ \left(3, \varepsilon, \left\{ \begin{array}{ll} \text{gen} & \mapsto (\text{gen}, \varepsilon) \\ \text{port} & \mapsto (\text{port}, \varepsilon) \end{array} \right.\right) \\ \left(2, id'_1, \left\{ \begin{array}{ll} \mathit{add} & \mapsto (\mathit{email}, \mathit{id}_1) \\ \mathit{info} & \mapsto (\mathit{data}, \mathit{id}_1) \end{array} \right.\right) \\ \left(2, id'_2, \left\{ \begin{array}{ll} \mathit{add} & \mapsto (\mathit{email}, \mathit{id}_2) \\ \mathit{info} & \mapsto (\mathit{data}, \mathit{id}_2) \end{array} \right.\right) \\ \left(5, \mathit{id}_2, \left\{ \text{gen} \mapsto (\text{gen}, \varepsilon) \right.\right) \end{array} \right\}$$

Marker properties

1. Marker allocation must be consistent:

   Two instances of the same process cannot be associated to the same marker during a computation sequence.

2. Marker allocation should be robust:

   Marker allocation should not depend on the interleaving order.

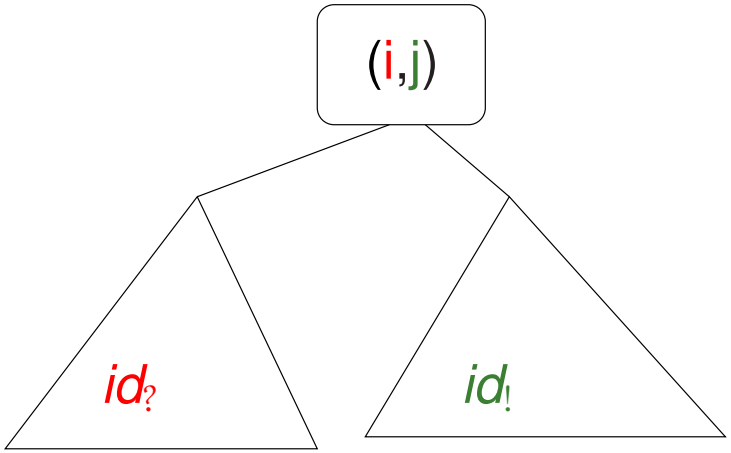
Marker allocation

Markers describe the history of the replications which have led to the creation of the threads.
They are binary trees:

- leaves are not labeled;
- nodes are labeled with a pair $(i, j) \in \mathit{Label}^2$.

They are recursively calculated when fetching resources as follows:

id_* :



Small step semantics

Small step semantics is given by a transition system:

- an initial configuration;
- three structural reduction rules which simulate the congruence relation;
- four action reduction rules which simulate the transition relation.

Initial configuration

$$C_0(\mathcal{S}) = (\mathcal{S}, \varepsilon, \emptyset)$$

Structural rules

$$C \cup \{(P \mid Q, id, E)\} \xrightarrow{\varepsilon} C \cup \{(P, id, E_{|fn(P)}); (Q, id, E_{|fn(Q)})\}$$

$$C \cup \{((\nu\ x)P, id, E)\} \xrightarrow{\varepsilon} C \cup \{(P, id, E[x \to (x, id)]_{|fn(P)})\}$$

$$C \cup \{(\emptyset, id, E)\} \xrightarrow{\varepsilon} C$$

Communication rules

$$\frac{E_?(y) = E_!(x)}{C \cup \left\{ \begin{array}{l} (y?^i[\overline{y}]P, id_?, E_?); \\ (x!^j[\overline{x}]Q, id_!, E_!) \end{array} \right\} \xrightarrow{(i,j)} C \cup \left\{ \begin{array}{l} (P, id_?, E_?[\overline{y} \rightarrow E_![\overline{x}]]_{|fn(P)}); \\ (Q, id_!, E_{!|fn(Q)}) \end{array} \right\}}$$

$$\frac{E_?(y) = E_!(x)}{C \cup \left\{ \begin{array}{l} (*y?^i[\overline{y}]P, id_?, E_?); \\ (x!^j[\overline{x}]Q, id_!, E_!) \end{array} \right\} \xrightarrow{(i,j)} C \cup \left\{ \begin{array}{l} (*y?^i[\overline{y}]P, id_?, E_?); \\ (P, N((i,j), id_?, id_!), E_?[\overline{y} \rightarrow E_!(\overline{x})]_{|fn(P)}); \\ (Q, id_!, E_{!|fn(Q)}) \end{array} \right\}}$$

Choice rules

$$C \cup \{P + Q, id, E\} \xrightarrow{\varepsilon} C \cup \{(P, id, E_{|fn(P)}\}$$
$$C \cup \{P + Q, id, E\} \xrightarrow{\varepsilon} C \cup \{(Q, id, E_{|fn(Q)}\}$$

Coherence

Theorem 1 Standard semantics and small step non-standard semantics are weakly bisimilar.

The main point is to prove that there are no conflicts between markers.

Marker allocation consistency

We denote by $\mathit{father}(\mathrm{P})$ the father of P, when it exists, in the syntactic tree of $\mathcal{S}$.

1. the thread $(\mathcal{S}, \varepsilon, \emptyset)$ can only be created at the start of the system computation;
2. a thread $(\mathrm{P}, \mathit{id}, \_)$ such that $\mathit{father}(\mathrm{P})$ is not a resource, can only be created by making a thread $(\mathit{father}(\mathrm{P}), \mathit{id}, \_)$ react;
3. a thread $(\mathrm{P}, N((\mathrm{i}, \mathrm{j}), \mathit{id}_?, \mathit{id}_!), \_)$ can only be created by making a thread $(\mathrm{P}_\mathrm{j}, \mathit{id}_!, \_)$ react (when P_j denote the syntactic process begining with the syntactic component labeled with j).

This proves marker allocation consistency.

Simplifying markers

We can simplify the shape of the marker without any loss of consistency:

1. replacing each tree by its right comb:

$$\begin{cases} \phi_1(N((i,j), id_1, id_2)) & = \phi_1(id_2).(i,j) \\ \phi_1(\varepsilon) & = \varepsilon \end{cases}$$

2. replacing pairs by their second component:

$$\begin{cases} \phi_2(N((i,j), id_1, id_2)) = \phi_2(id_2).j \\ \phi_2(\varepsilon) = \varepsilon \end{cases}$$

Those simplifications can be seen as an abstraction, they do not loose semantics consistency, but they may abstract away information, in the case of nested resources, by merging information about distinct computation sequences.

Middle semantics

Small step semantics can be analyzed but:

- there are too many transition rules;
- it uses too many kinds of processes.

$\Longrightarrow$ We design a new semantics with only active rules.

(Structural rules are included inside active rules)

Definition

Structural rules:

$$C \cup \{(P \mid Q, id, E)\} \xrightarrow{\varepsilon} C \cup \{(P, id, E_{|fn(P)}); (Q, id, E_{|fn(Q)})\}$$

$$C \cup \{((\nu\ x)P, id, E)\} \xrightarrow{\varepsilon} C \cup \{(P, id, E[x \to (x, id)]_{|fn(P)})\}$$

$$C \cup \{(\emptyset, id, E)\} \xrightarrow{\varepsilon} C$$

are a confluent and well-founded transition system,
we denote by $\Longrightarrow$ its limit:

$$a \Longrightarrow b \textit{ ssi } \begin{cases} a \to^* b \\ \forall c, b \not\to c. \end{cases}$$

and we define our new transition system by $\xrightarrow{\lambda}{}' = \xrightarrow{\lambda} \circ \Longrightarrow$.

Extraction function

An extraction function calculates the set of the thread instances spawned at the beginning of the system execution or after a computation step.

$$
\begin{array}{rcl}
\beta((\nu\ n)P, id, E) & = & \beta(P, id, (E[n \mapsto (n, id)])) \\
\beta(\emptyset, id, E) & = & \emptyset \\
\beta(P \mid Q, id, E) & = & \beta(P, id, E) \cup \beta(Q, id, E) \\
\beta(P+Q, id, E) & = & \{(P+Q, id, E_{|fn(P+Q)})\} \\
\beta(y?^{i}[\overline{y}].P, id, E) & = & \{(y?^{i}[\overline{y}].P, id, E_{|fn(y?^{i}[\overline{y}].P)})\} \\
\beta(*y?^{i}[\overline{y}].P, id, E) & = & \{(*y?^{i}[\overline{y}].P, id, E_{|fn(*y?^{i}[\overline{y}].P)})\} \\
\beta(x!^{j}[\overline{x}].P, id, E) & = & \{(x!^{j}[\overline{x}].P, id, E_{|fn(x!^{j}[\overline{x}]P)})\}
\end{array}
$$

Transition system

$$C_0(\mathcal{S}) = \beta(\mathcal{S}, \varepsilon, \emptyset)$$

$$C \cup \{(P+Q, id, E)\} \xrightarrow{\varepsilon} (C \cup \beta(P, id, E))$$
$$C \cup \{(P+Q, id, E)\} \xrightarrow{\varepsilon} (C \cup \beta(Q, id, E))$$

Communication rules

$$\frac{E_?(y) = E_!(x)}{C \cup \left\{ \begin{array}{l} (y?^i[\overline{y}]P, id_?, E_?), \\ (x!^j[\overline{x}]Q, id_!, E_!) \end{array} \right\} \xrightarrow{(i,j)} (C \cup \beta(P, id_?, E_?[y_i \mapsto E_!(x_i)]) \cup \beta(Q, id_!, E_!))}$$

$$\frac{E_*(y) = E_!(x)}{C \cup \left\{ \begin{array}{l} (*y?^i[\overline{y}]P, id_*, E_*), \\ (x!^j[\overline{x}]Q, id_!, E_!) \end{array} \right\} \xrightarrow{(i,j)} \left(\begin{array}{l} C \cup \{(*y?^i[\overline{y}]P, id_*, E_*)\} \\ \quad \cup \beta(P, N((i,j), id_*, id_!), E_*[y_i \mapsto E_!(x_i)]) \\ \quad \cup \beta(Q, id_!, E_!) \end{array} \right)}$$

Coherence

Middle semantics and standard semantics are strongly bisimilar, but we still consider too much process: we can also factor choice operations.

For that purpose we restrict our study to the computation sequences in where communication are only made when there are no choice thread instance at top level, and factor choices with communication rules.

Definition

Choice rules are a well-founded transition system,

we denote by $\Longrightarrow$ its non-deterministic limit:

$$a \Longrightarrow b \textit{ ssi } \begin{cases} a \rightarrow^* b \\ \forall c, b \not\rightarrow c. \end{cases}$$

and we define our new transition system by $\overset{\lambda}{\rightarrow}' = \overset{\lambda}{\rightarrow} \circ \Longrightarrow$.

Extraction function

An extraction function calculates the set of all choices for the set of the thread instances spawned at the beginning of the system execution or after a communication.

$$
\begin{aligned}
\beta((\nu\ n)P, id, E) &= \beta(P, id, (E[n \mapsto (n, id)])) \\
\beta(\emptyset, id, E) &= \{\emptyset\} \\
\beta(P+Q, id, E) &= \beta(P, id, E) \cup \beta(Q, id, E) \\
\beta(P \mid Q, id, E) &= \{A \cup B \mid A \in \beta(P, id, E),\ B \in \beta(Q, id, E)\} \\
\beta(y?^i[\overline{y}].P, id, E) &= \{\{(y?^i[\overline{y}].P, id, E_{|fn(y?^i[\overline{y}].P)})\}\} \\
\beta(*y?^i[\overline{y}].P, id, E) &= \{\{(*y?^i[\overline{y}].P, id, E_{|fn(*y?^i[\overline{y}].P)})\}\} \\
\beta(x!^j[\overline{x}].P, id, E) &= \{\{(x!^j[\overline{x}].P, id, E_{|fn(x!^j[\overline{x}]P)})\}\}
\end{aligned}
$$

Transition system

$$C_0(\mathcal{S}) = \beta(\mathcal{S}, \varepsilon, \emptyset)$$

$$\frac{E_?(y) = E_!(x),\ \mathit{Cont}_P \in \beta(P, id_?, E_?[y_i \mapsto E_!(x_i)]),\ \mathit{Cont}_Q \in \beta(Q, id_!, E_!)}{C \cup \{(y?^i[\overline{y}]P, id_?, E_?), (x!^j[\overline{x}]Q, id_!, E_!)\} \xrightarrow{(i,j)} (C \cup \mathit{Cont}_P \cup \mathit{Cont}_Q)}$$

$$\frac{E_*(y) = E_!(x),\ \mathit{Cont}_P \in \beta(P, N((i,j), id_*, id_!), E_*[y_i \mapsto E_!(x_i)]),\ \mathit{Cont}_Q \in \beta(Q, id_!, E_!)}{C \cup \left\{ \begin{array}{l} (*y?^i[\overline{y}]P, id_*, E_*), \\ (x!^j[\overline{x}]Q, id_!, E_!) \end{array} \right\} \xrightarrow{(i,j)} \left(C \cup \{(*y?^i[\overline{y}]P, id_*, E_*)\} \cup \mathit{Cont}_P \cup \mathit{Cont}_Q\right)}$$

META-language: intuition

In the π-calculus :

- each program point $a?[y]P$ is associated with a partial interaction:

$$(in, [a], [y], label(P))$$

- each program point $b![x]Q$ is associated with a partial interaction:

$$(out, [b, x], [], label(Q))$$

- The generic transition rule:

$$((in, out), [X_1^1 = X_1^2], [Y_1^1 \leftarrow X_2^2]).$$

  describes communication steps.

Some rules are more complex (e.g. ambient opening).

Advantages of the META-language

1. each analysis at the META-language level provides an analysis for each encoded model;
2. the META-language avoids the use of congruence and α-conversion:

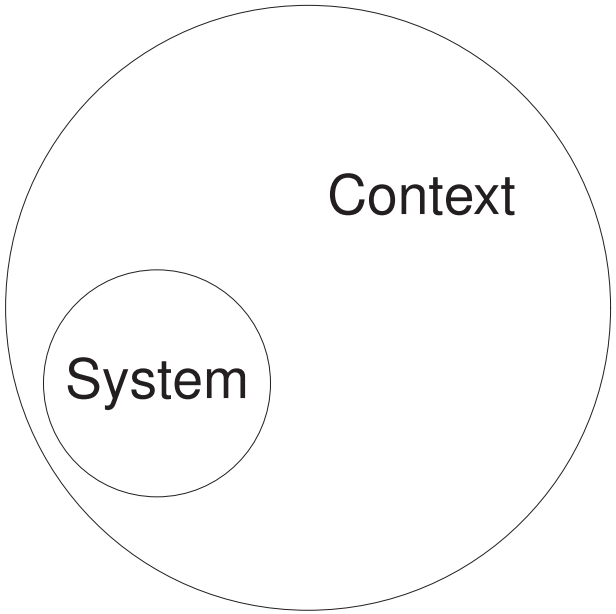
   Fresh names are allocated according to the local history of each process.
3. names contains useful information:

   This allows the inference of:

   - more complex properties;
   - some simple properties the proof of which uses complex properties.

Context-free analysis

Analyzing interaction between a system and its unknown context.



The context may

- spy the system, by listening to message on unsafe channel names;
- spoil the system, by sending message via unsafe channel names.

Nasty context

Context := (ν unsafe) (**new**
| **spy**$_0$ | ... | **spy**$_n$
| **spoil**$_0$ | ... | **spoil**$_n$)

where

new := ($*$(ν *channel*)$*$unsafe![*channel*])

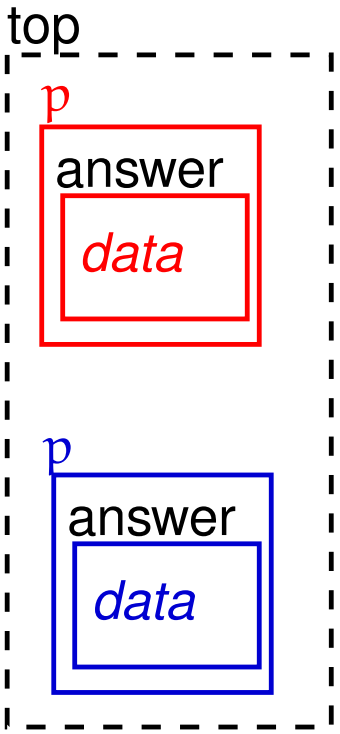
spoil$_k$:= ($*$unsafe?[c]unsafe?[x_1]...unsafe?[x_k]c![x_1,...,x_k])

spy$_k$:= ($*$unsafe?[c]c?[x_1,...,x_k] (($*$unsafe![x_1]) | ... | ($*$unsafe![x_k])))

Non-Standard Configuration

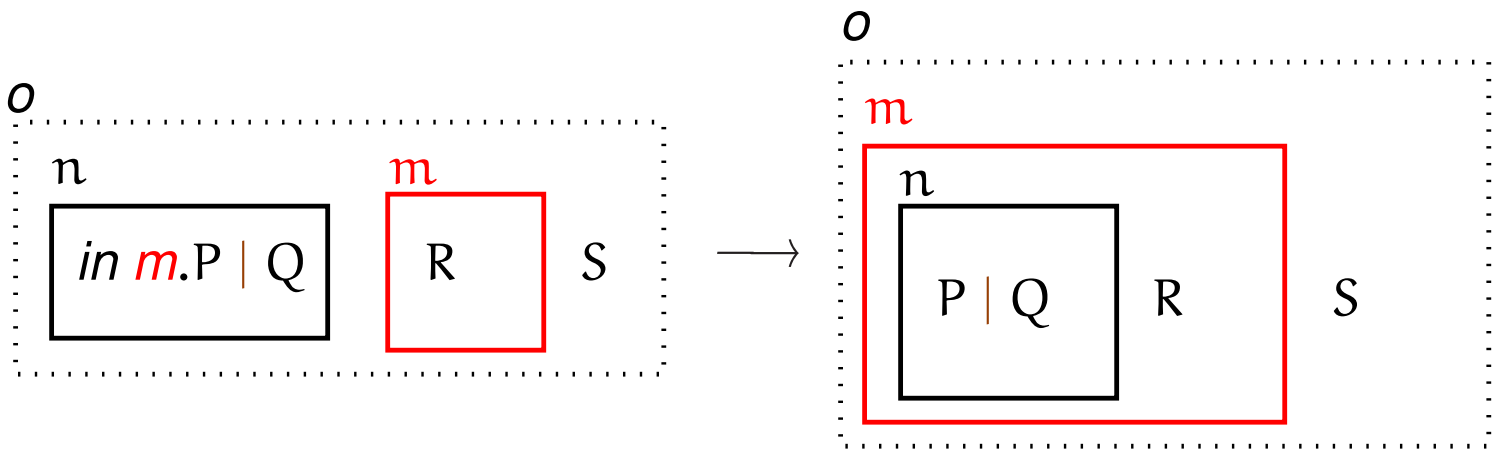
We flatly represent system configurations:

$$\left\{\begin{array}{l}(\mathrm{p}^{12}[\bullet], id_0, (\mathrm{top}, \varepsilon), [\mathrm{p} \mapsto (\mathrm{p}, id_0)]) \\ (\mathrm{p}^{12}[\bullet], id_1, (\mathrm{top}, \varepsilon), [\mathrm{p} \mapsto (\mathrm{p}, id_1)]) \\ (\mathrm{answer}^{8}[\bullet], id'_0, (12, id_0), \emptyset) \\ (\mathrm{answer}^{8}[\bullet], id'_1, (12, id_1), \emptyset) \\ \hline (\langle rep \rangle^{9}, id'_0, (8, id'_0), [rep \mapsto (data, id_0)]) \\ (\langle rep \rangle^{9}, id'_1, (8, id'_1), [rep \mapsto (data, id_1)])\end{array}\right.$$



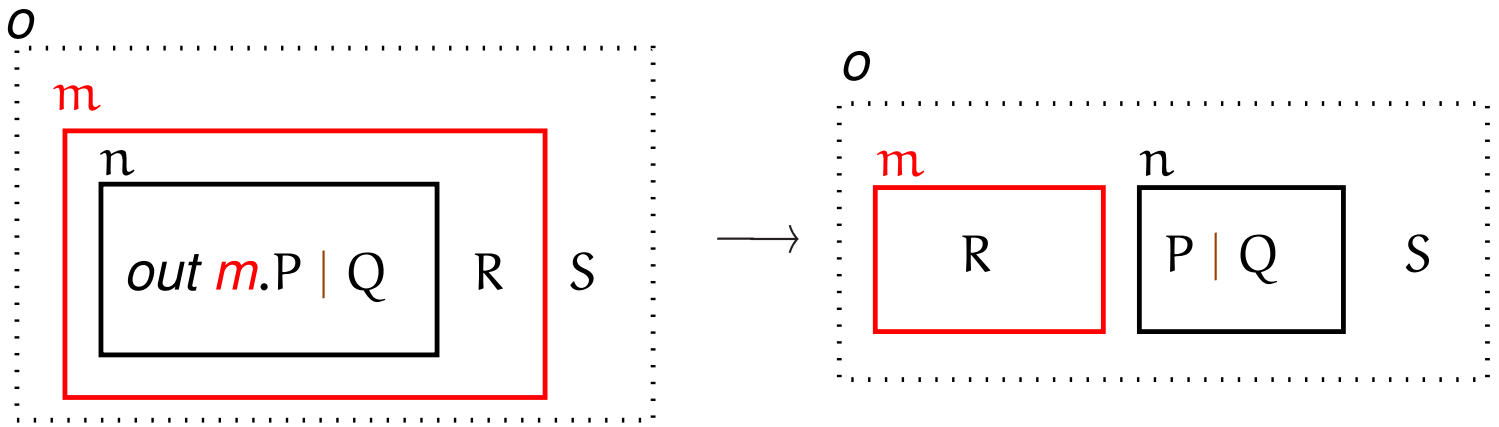
In migration

$$\frac{\begin{cases} \lambda = \left(n^i[\bullet], id_1, loc_1, E_1\right), \\ \mu = \left(m^j[\bullet], id_2, loc_2, E_2\right), \\ \psi = \left(in^k o.P, id_3, loc_3, E_3\right), \\ loc_1 = loc_2,\ loc_3 = (i, id_1),\ E_2(m) = E_3(o),\ \lambda \neq \mu. \end{cases}}{C \cup \{\lambda; \mu; \psi\} \overset{in(i,j,k)}{\longrightarrow} (C \cup \{\mu\}) \cup (n^i[\bullet], id_1, (j, id_2), E_1) \cup \beta\left(P, id_3, loc_3, E_{3|fn(P)}\right).}$$



out migration

$$\frac{\begin{cases} \lambda = \left(m^{i}[\bullet], id_1, loc_1, E_1\right), \\ \mu = \left(n^{j}[\bullet], id_2, loc_2, E_2\right), \\ \psi = \left(out^{k} o.P, id_3, loc_3, E_3\right), \\ loc_2 = (i, id_1),\ loc_3 = (j, id_2),\ E_1(m) = E_3(o) \end{cases}}{C \cup \{\lambda; \mu; \psi\} \xrightarrow{out(i,j,k)} (C \cup \{\lambda\}) \cup (n^{j}[\bullet], id_2, loc_1, E_2) \cup \beta\left(P, id_3, loc_3, E_{3|fn(P)}\right).}$$



Dissolution

$$\frac{\begin{cases} \lambda = \left(\mathit{open}^{i} m.P, \mathit{id}_1, \mathit{loc}_1, E_1\right) \\ \mu = \left(n^{j}[\bullet], \mathit{id}_2, \mathit{loc}_2, E_2\right), \\ \mathit{loc}_1 = \mathit{loc}_2,\ E_1(m) = E_2(n), \end{cases}}{C \cup \{\lambda; \mu\} \xrightarrow{\mathit{open}(i,j)} (C \setminus A) \cup A' \cup \beta\left(P, \mathit{id}_1, \mathit{loc}_1, E_{1|fn(P)}\right)}$$

$$\text{where} \begin{cases} A = \{(a, \mathit{id}, \mathit{loc}, E) \in C \mid \mathit{loc} = (j, \mathit{id}_2)\} \\ A' = \{(a, \mathit{id}, \mathit{loc}_2, E) \mid (a, \mathit{id}, (j, \mathit{id}_2), E) \in C\}. \end{cases}$$